Amendments to the Claims: This listing of claims will replace all prior versions, and listings, of claims in the application

Listing of Claims:

1. (Currently Amended) A method of task management using memory ranges of shared memory, comprising the steps of:

a. executing multiple instances of a kernel;

b. ~~receiving~~generating one or more tasks to be executed from a plurality of the instances of the kernel;

~~b~~c. atomizing the one or more tasks into one or more atomic sub-tasks; and

~~c~~d. assigning protection attributes indicating a portion of one of the memory ranges of ~~a~~the shared memory for each respective atomic sub-task of the one or more atomic sub-tasks such that each respective sub-task is executed by one of a plurality of processors which inherits access rights to the shared memory indicated by the protection attributes corresponding to the respective atomic sub-task,

wherein each of the protection attributes corresponding to a common instance of the multiple instances of the kernel is assigned such that the inherited access rights of the one or more processors which relate to respective sub-tasks derived from a respectively different common instance corresponds to a respectively different one of the memory ranges of the shared memory.

2. (Original) The method according to claim 1, further comprising the step of:

d. scheduling the one or more atomic sub-tasks into a central task queue.

3. (Previously Presented) The method according to claim 2, wherein the step of scheduling the one or more atomic sub-tasks into the central task queue is done according to one or both of temporal and priority considerations.

4. (Previously Presented) The method according to claim 2, further comprising the step of:

e. obtaining from a first idle processor of the plurality of processors a first atomic sub-task from the central task queue, the first idle processor thereby inheriting the access rights to a first memory range of the shared memory in executing the first atomic sub-task.

5. (Previously Presented) The method according to claim 4, further comprising the step of:

f. obtaining from a further idle processor of the plurality of processors a further atomic sub-task from the central task queue, the further idle processor thereby inheriting the access rights to a further memory range of the shared memory in executing the further atomic sub-task.

6. (Original) The method according to claim 5, wherein steps e and f are repeated until there are no further idle processors or no further atomic sub-tasks in the central task queue.

7. (Original) The method according to claim 6, further comprising the step of combining one or more atomic results of execution of each atomic sub-task corresponding to a task into a result of the task.

8. (Currently Amended) A task-based library for processor management for execution of multiple instances of a kernel configured to use memory ranges of shared memory, comprising:

means for receiving tasks to be executed from the multiple instances of the kernel;

a task atomizer for atomizing the tasks into one or more atomic sub-tasks;

a plurality of processors for executing the one or more atomic sub-tasks; and

an access rights generator for assigning protection attributes indicating a portion of one of the memory ranges of athe shared memory for each respective atomic sub-task of the one or more atomic sub-tasks such that each respective sub-task is executed by one of the plurality of processors which inherits access rights to the shared memory indicated by the protection attributes corresponding to the respective atomic sub-task, wherein each of the protection attributes corresponding to a common instance of the multiple instances of the kernel is assigned such that the inherited access rights of the one or more processors which relate to respective sub-tasks derived from a respectively different common instance corresponds to a respectively different one of the memory ranges of the shared memory.

9. (Original) The task-based library of claim 8, further comprising a central task queue for storing the one or more atomic sub-tasks waiting to be executed.

10. (Original) The task-based library of claim 9, further comprising a task scheduler for arranging the one or more atomic sub-tasks in the central task queue.

11. (Original) The task-based library of claim 10, further comprising a combiner for combining execution results of the one or more atomic sub-tasks into an execution result of a task.

12. (Canceled)

13. (Original) The task-based library of claim 8, further comprising a combiner for combining execution results of the one or more atomic sub-tasks into an execution result of a task.

14-15. (Canceled)

16. (Currently Amended) A method of task management for execution of multiple instances of a kernel using memory ranges of shared memory, comprising the steps of:

a. receiving one or more tasks to be executed from the multiple instances of the kernel;

b. atomizing the one or more tasks into one or more atomic sub-tasks;

c. assigning protection attributes indicating a portion of one of the memory ranges of athe shared memory for each respective atomic sub-task of the one or more atomic sub-tasks such that each respective sub-task is executed by an idle processor of a plurality of processors which inherits access rights to the shared memory indicated by the protection attributes corresponding to the respective atomic sub-task;

d. scheduling the one or more atomic sub-tasks into a central task queue according to one or both of temporal and priority considerations;

e. obtaining via a first idle processor of the plurality of processors a first atomic sub-task from the central task queue for execution of the first atomic sub-task; and

f. obtaining via a further idle processor of the plurality of processors a further atomic sub-task from the central task queue for execution of the further atomic sub-task,

wherein each of the protection attributes corresponding to a common instance of the multiple instances of the kernel is assigned such that the inherited access rights of the one or more processors which relate to respective sub-tasks derived from a respectively different common instance corresponds to a respectively different one of the memory ranges of the shared memory.

17. (Original) The method according to claim 16, further comprising the step of repeating steps e and f until there are no further idle processors or no further atomic sub-tasks in the central task queue.

18. (Original) The method according to claim 17, further comprising the step of combining one or more atomic results of execution of each atomic sub-task corresponding to a task into a result of the task.

19. (Previously Presented) The method according to claim 16, wherein the step d of scheduling the one or more atomic sub-tasks into the central task queue is done according to temporal considerations.

20. (Previously Presented) The method according to claim 16, wherein the step d of scheduling the one or more atomic sub-tasks into the central task queue is done according to priority considerations.

21. (New) The method according to claim 1, further comprising

designating a master kernel;

submitting, by the multiple instances of the kernel, the one or more atomic sub-tasks to the master kernel;

scheduling, by the master kernel, all of the multiple other instances of the kernel;

consolidating priority and temporal execution parameters of each sub-task; and

placing the one or more sub-tasks into a central task-queue after the consolidating step.

22. (New) The method according to claim 21, further comprising:

determining whether any of the plurality of processors are idle;

responsive to one of the processors being idle, receiving, by the processor determined to be idle, a first atomic sub-task using, the shared memory designated by the corresponding protection attribute; and

repeatedly and simultaneously determining whether another processor is idle and executing a subsequent atomic sub-task until all tasks are completed.

23. (New) The method of claim 22, further comprising

providing a summing junction as part of the kernel;

combining the execution results of each of the atomic sub-tasks for a completed task using the summing junction; and

routing the combined execution results of the completed task to an input/output port for delivery to a calling process.